Manual básico de uso para crear proyectos base en Django

Lubuntu + Python

Contenido:

Comandos esenciales de terminal
Fragmentos de código Python
Consejos prácticos

8 de abril de 2026

Autor: **Jonathan Calvopiña**
Basado en Lubuntu 22.04 LTS

1. Comandos de terminal (Linux / Lubuntu)

Comandos para crear proyecto Django

Primero abrimos la consola o el terminal, en mi caso utilizo lubuntu, pero funciona en cualquier linux con sus derivados de Ubuntu.

```
m@m:~ $
```

Comandos básicos para crear la carpeta del proyectoN, primero nos dirigimos a la carpeta en la que queremos guardar y luego ejecutamos.

```
m@m:~ $cd Documentos/python
m@m:~ $/Documentos/python $ mkdir -p carpetaN
m@m:~ $/Documentos/python $ cd carpetaN
m@m:~ $/Documentos/python/carpetaN $
```

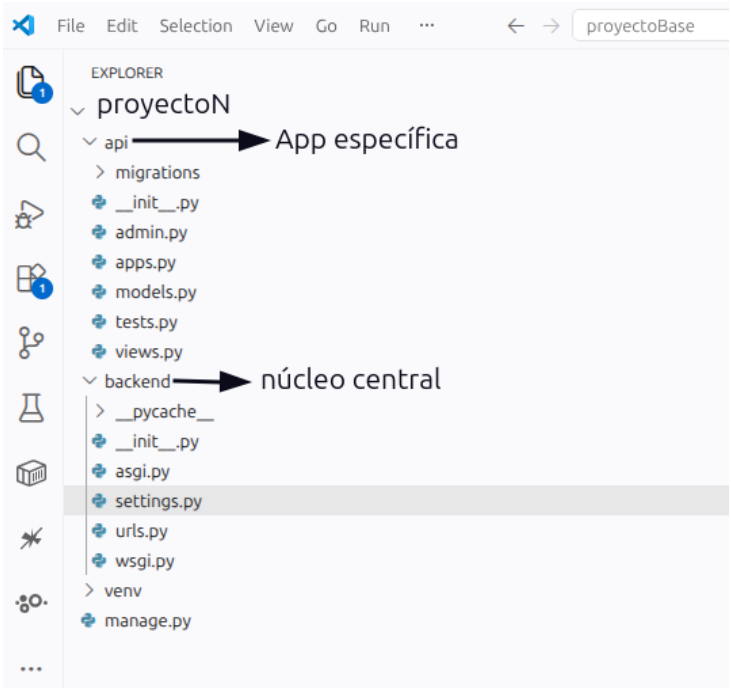
Ahora vamos a crear el protecto django, es decir el backend donde va el **núcleo central**, este viene ha ser la infraestructura principal.

```
m@m:~ $/Documentos/python/carpetaN $ python3 -m venv venv
m@m:~ $(venv) /Documentos/python/carpetaN $ pip install django
m@m:~ $(venv) /.../carpetaN $ django-admin startproject backend .
```

Luego creamos el administrador donde va todos los cambios del sistema llamado api; es decir, donde creamos la **app específica**.

```
m@m:~ $(venv) /.../carpetaN$ python ../manage.py startapp api
m@m:~ $(venv) /.../carpetaN/$
```

Figura 1: Proyecto Django en Visual Studio Code.



Fuente: *Elaboración propia.*

Gestión de archivos python

Abrimos **backend/settings.py** en VS Code y buscamos la sección **INSTALLED_APPS**.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'api',  #     AGREGA ESTA LINEA AL FINAL
]
```

Creamos un archivo **urls.py** dentro de la carpeta **backend** y luego vamos a la dirección **backend/urls.py** y agregamos lo siguiente.

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')),# Conecta con la app api
]
```

Borramos todo y pegamos lo siguiente en **api/urls.py**.

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

Abrimos **api/views.py** y pegamos:

```
from django.http import JsonResponse

def index(request):
    return JsonResponse({"mensaje": "API funcionando"})
```

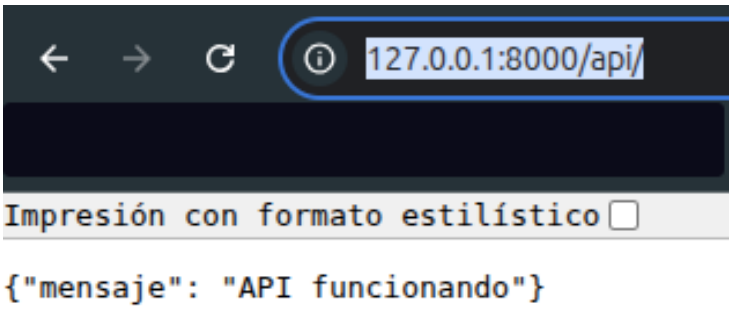
Realizamos la introducción del siguiente código para que corra Django en la web.

```
m@m:~ $(venv) /.../carpetaN$ python manage.py migrate
m@m:~ $(venv) /.../carpetaN/$ python manage.py runserver

Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 08, 2026 - 03:13:33
Django version 6.0.4, using settings 'backend.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

WARNING: This is a development server. Do not use it in a production setting.
   Use a production WSGI or ASGI server instead.
```

```
For more information on production servers see: https://docs.djangoproject.com/
    en/6.0/howto/deployment/
```

copiamos el siguiente enlace `http://127.0.0.1:8000/api/` y lo pegamos en el navegador web, sea Google, etc. Al buscar, debería aparecer como la imagen a continuación:

Figura 2: Navegador web.



Fuente: *Elaboración propia.*

Vamos a crear una carpeta dentro de api llamado **templates** donde va todo documento html, además una subcarpeta **api** y dentro de esta el archivo **index.html** que vayamos a desarrollar en nuestra página web, lo que es el frontend.

Figura 3: proyectoN/api/templates/api/index.html.

Fuente: *Elaboración propia.*

Agregamos funciones python

Vamos a crear otra función en **api/views.py**; en este caso creamos la función análisis.

```
from django.http import JsonResponse
def index(request):
    return JsonResponse({"mensaje": "API funcionando"})

def analisis(request):
```

```
    datos = {"media": 25.5, "desviacion": 4.2}
    return JsonResponse(datos)
```

Agregamos en la dirección **api/urls.py**; en este caso creamos de la función análisis.

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    # http://127.0.0.1:8000/api/
    path('analisis/', views.analisis, name='analisis'),
    # http://127.0.0.1:8000/api/analisis/
]
```

Conectamos con una BD PostgreSQL

Instalamos por el terminal la biblioteca:

```
m@m:~ $(venv) /.../carpetaN/$ pip install psycopg2-binary
```

Modificamos en **backend/urls.py** buscamos la sección DATABASES y remplazamos por:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'MiBaseDatos',  # Nombre de BD
        'USER': 'postgres',  # Usuario PostgreSQL
        'PASSWORD': 'tu_contrase a', # contrase a de PostgreSQL
        'HOST': 'localhost', # IP del servidor
        'PORT': '5432',  # Puerto por defecto de PostgreSQL
    }
}
```

Verificar conexión:

```
m@m:~ $python manage.py migrate
m@m:~ $python manage.py runserver
```

Salimos presionando Ctrl+C, luego creamos el superusuario, donde te solicita: usuario: nombre que tu prefieras. email: tucorreo@hotmail.com. password: tu contraseña para ingresar a este proyecto.

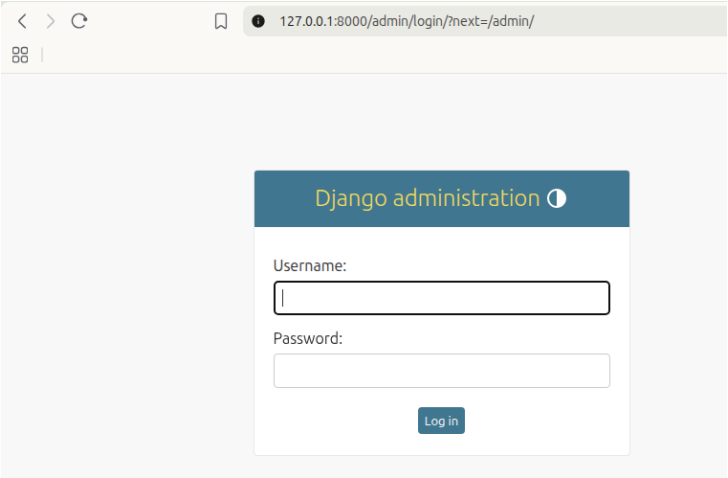
```
m@m:~ $python manage.py createsuperuser
m@m:~ $# Ingresar: usuario, email, password
```

Y por que es importante un superusuario,nos permite facilitar el acceso al administrador donde tenemos nuestras bases de datos. Ejecutamos por la terminal.

```
m@m:~ $python manage.py runserver
```

Ingresamos en el navegador lo siguiente: `http://127.0.0.1:8000/admin/` y debe mostrarnos la ventana donde nos solicita usuario y contraseña:

Figura 4: Página de ingreso de administrador.



Fuente: *Elaboración propia.*

Por otra parte, podemos crear bases de datos directo en ORM, en vez de SQL puro, modificando **api/admin.py**, como por ejemplo:

```
from django.contrib import admin
from .models import Persona

@admin.register(Persona)
class PersonaAdmin(admin.ModelAdmin):
    list_display = ['id', 'nombre', 'edad', 'coordenadax', '
       coordenaday']
    search_fields = ['nombre']
    list_filter = ['edad']
```

Luego modificamos **api/models.py**, por ejemplo:

```
# models.py - Es como definir las columnas del Excel
from django.db import models

class Persona(models.Model):
    # Django crea automatico: id = 1, 2, 3... (nunca se repite)
    nombre = models.CharField(max_length=100) # Texto, max 100
       letras
    edad = models.IntegerField() # Cantidad entera
    coordenadax = models.DecimalField(max_digits=10,
       decimal_places=6)  # Decimales
    coordenaday = models.DecimalField(max_digits=10,
       decimal_places=6)

    def __str__(self):
        return f"{self.nombre} ({self.edad} edad)"
```

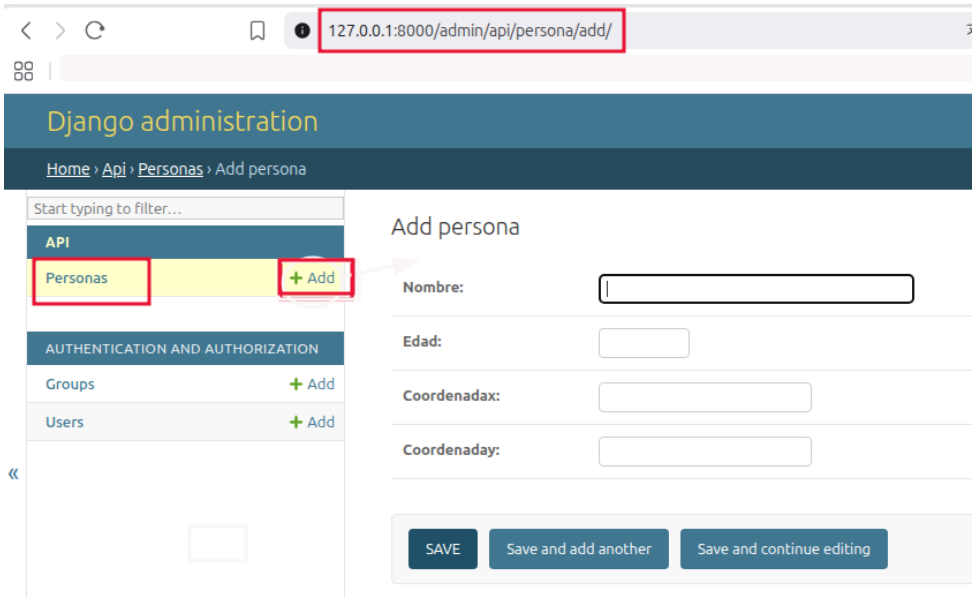
Ingresamos por consola:

```
m@m:~ $# 1. Crear migracin
m@m:~ $python manage.py makemigrations
m@m:~ $# 2. Aplicar a base de datos
m@m:~ $python manage.py migrate
```

```
m@m:~ $# Reinicia el servidor con Ctrl+C
m@m:~ $python manage.py runserver
```

Al dar en aceptar, nos redirigira al `http://127.0.0.1:8000/admin/api/persona/add/` y debe mostrarnos la ventana donde nos solicita usuario y contraseña:

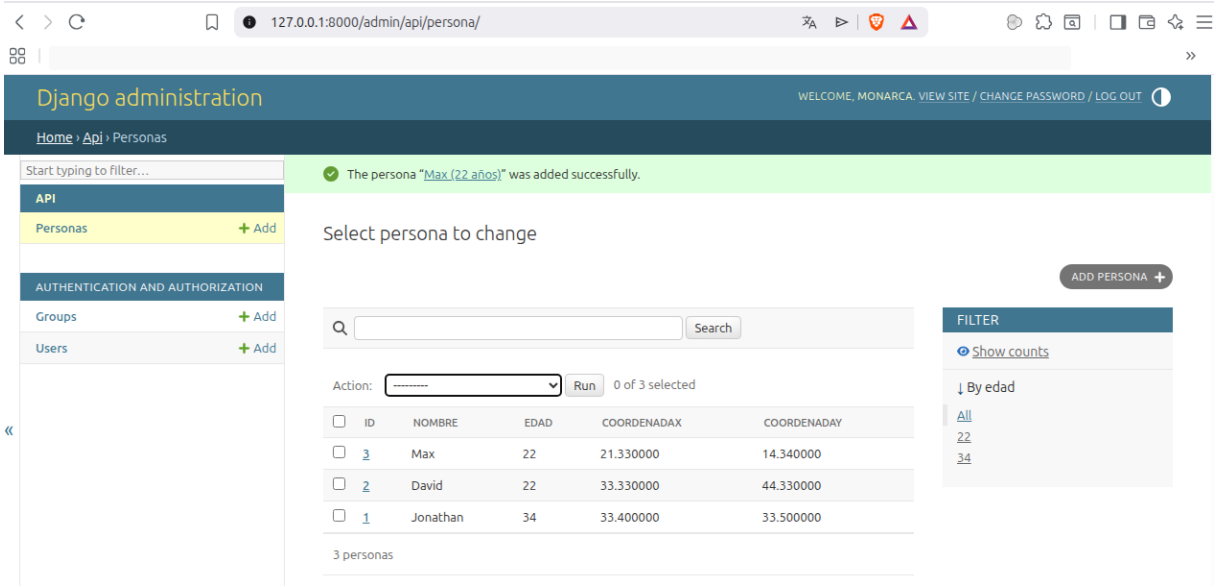
Figura 5: Página de ingreso de personas.



Fuente: *Elaboración propia.*

Para añadir damos clic en el simbolo más add e ingresamos los datos solicitados, y damos clic en guardar.

Figura 6: Página tabla de datos.



Fuente: *Elaboración propia.*